LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-677840

A Novel Vertex Affinity for Community Detection

A. Yoo, G. Sanders, V. Henson, P. Vassilevski

October 5, 2015

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

A Novel Vertex Affinity for Community Detection*

Andy Yoo, Geoffrey Sanders, Van Henson, and Panayot Vassilevski

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94550

Abstract

We propose a novel vertex affinity measure in this paper. The new vertex affinity quantifies the proximity between two vertices in terms of their clustering strength and is ideal for such graph analytics applications as community detection. We also developed a framework that combines simple graph searches and resistance circuit formulas to compute the vertex affinity efficiently. We study the properties of the new affinity measure empirically in comparison to those of other popular vertex proximity metrics. Our results show that the existing metrics are ill-suited for community detection due to their lack of fundamental properties that are essential for correctly capturing inter- and intra-cluster vertex proximity.

1 Introduction

Graphs have been used as an effective means to explore the relations between data object. Graph analytics has gained a great deal of interest in recent years as the demand for the capability to mine useful information from an increasingly large volume of data grew stronger. The graph analytics has found its uses in a wide range of applications, such as social network analysis, web mining, bioinformatics, and social media analysis.

One of the fundamental and frequently utilized functionalities in graph analytics is to measure how *close* two vertices in given graph are to each other. Many metrics that measure the distance between vertices in a graph are reported in literature [19, 24, 3, 26, 21, 10, 7] and are applied to solving a variety of graph analytics applications [35, 37, 26, 14, 25, 4].

These metrics commonly rely on the paths between vertices to determine the proximity between them. However, they differ in how to combine the path information in their computation and hence compute closely related, but slightly different proximity measures. As a result, the outcome from the same graph analytics application may vary quite considerably depending on the type of the proximity metrics used. For example, the commute time [21] renders very high degree vertices to be closer to other vertices, whereas the Katz score [19] places vertices that are connected by a large number of short paths closer to each other. Therefore, it is critical to use vertex proximity metrics that are best suited for intended applications. By the same token, for any proximity metric to be effective, it must be defined within the context of target applications.

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

We propose a new vertex proximity concept called *affinity* for community detection [23] in this paper. Community detection refers to a graph analytics problem to find a set of densely-connected subgraphs, each of which represents a community structure in a graph. Community detection is an important graph analytics application that has found its uses in a wide spectrum of applications, including social network analysis, bioinformatics, biology, social media analysis, and web mining, and has received a great deal of attention recently [13, 16, 29, 11, 18, 9, 12, 1, 8, 6, 17]. To cluster vertices that are densely-linked to each other effectively, we define the affinity as a measure of proximity between vertices in terms of their *clustering strength*. That is, our affinity indicates how close two vertices are to each other in terms of their possibility of being in the same community.

The affinity measure is designed based on two basic ideas. Since vertices in the same community are highly likely to be connected to each other by direct edges, there are expected to be many short paths connecting any two vertices in the community. Therefore, we argue that the clustering strength of two vertices should increase if they are connected by a large number of short paths. We also argue that the affinity between two vertices, u and v, is also affected by the affinity among the vertices on the paths that connect u and v. This is because if two vertices are in the same community, it is likely that the vertices on the paths connecting them are connected by many short paths as well.

We developed an efficient affinity computation framework, where we implement these ideas by combining simple path searches and resistance circuit formulas. In our framework, a weight is first assigned to each edge, which reflects the clustering strength of two endpoints of the edge. Given two vertices, it finds a set of independent shortest paths connecting them through a sequence of shortest (weighted) path searches. Finally, affinity is computed via simple resistance circuit formulas, modeling the set of shortest paths found as resistance circuit. An advantage of the developed framework, in addition to its computational efficiency, is that once edges are assigned with application-specific weights, then the same framework can be used for a wide range of applications.

Another main objective of this research is to empirically study the properties of the new affinity measure in comparison with those of existing proximity metrics to evaluate their effectiveness as a means for community detection. The results from this study show that the existing metrics are not suitable for community detection as they lack some fundamental properties that are essential for correctly capturing inter- and intra-community vertex proximity.

The paper is organized as follows. We provide preliminaries in Section 2, and related work are discussed in Section 3. The new affinity measure and its computation framework are described in Section 4. The results from the empirical study are discussed in Section 5. Concluding remarks and directions for future work are given in Section 6.

2 Preliminaries

Graph $G = (V, E)$ consists of sets of vertices and edges, denoted by V and E, respectively. Given an edge $e = (u, v) \in E$, vertex v is said to be *adjacent* to vertex u and vertices u and v are said to be *incident* to the edge e (an edge $e = (u, v)$ is also denoted by $e(u, v)$). A set of vertices that are adjacent to vertex u is called the *neighbors* of u and denoted by N_u. If each edge represents an ordered pair of vertices, the graph is called *directed*. Otherwise, it is called *undirected*. A graph is *complete* if any two vertices in the graph are connected by an edge.

The vertices and edges also can have weights, where the weight of a vertex v and an edge $e(i,j)$ are denoted by w_v and w_{ij}, respectively. A graph that has weighted edges is called *weighted graph*. An *unweighted* (or *binary*) graph is a graph with edges with the unit weight of 1.

A *path* from a *source* vertex s to a *destination* vertex t is a sequence $<v_0, v_1, \ldots, v_k>$ such that $s = v_0$, $t = v_k$ and $(v_{i-1}, v_i) \in E$, for $i = 1, 2, \ldots, k$ and is denoted by $p(s,t)$, The *length* of the path for an unweighted graph is the number of edges in the path (k). For a weighted graph, the length is the sum of the weight of edges in the path. Two paths are *independent* if they do not share common vertices except the source and destination vertices. *Minimum independent shortest paths* between s and t refer to a set of all possible independent shortest paths between s and t that gives the minimum total weight sum.

Graphs, either weighted or unweighted, can be represented as a matrix. The *adjacency matrix* of a given graph $G = (V,E)$ is a $|V| \times |V|$ matrix $M = (m_{ij})$ such that $m_{ij} = w_{ij}$ if $(i,j) \in E$ and $m_{ij} = 0$ otherwise. By $d_i = \sum_{j=1}^{|V|} w_{ij}$, we denote the *degree* of a vertex i. The *diagonal matrix* is a $|V| \times |V|$ matrix, such that $D_{ii} = d_i$ and $D_{ij} = 0$ for $i \neq j$. The *volume* of the matrix $V(M) = \sum_{i=1}^{|V|} d_i$. Given M and D, the (unnormalized) *graph Laplacian* of the graph is $L = D - M$.

3 Related Work

Various metrics are currently used to measure the proximity between vertices in a graph. The simplest metric among these is the geodesic distance that is defined as the length of shortest path between two given vertices. The geodesic distance between two vertices can be found by running simple breadth-first search [10]. There are a set of metrics that consider the neighbors of given vertices in measuring the proximity, such as *common neighbors* metric [22], *Jaccard's coefficient* [31], and *Adamic's* metric [2].

As opposed to the shortest path distance and neighbor-based metrics, there are a group of metrics that take into account all the paths between given pair of vertices. In general, the value of these metrics decreases as the number of paths connecting the vertices increases and their average length decreases.

The *Katz score* [19] measures the vertex proximity by combining the number as well as the length of the paths. It is defined by the following equation

$$Katz(s,t) = \sum_{l=1}^{\infty} \alpha^l |paths_{s,t}^{<l>}|, \tag{1}$$

where $paths_{s,t}^{<l>}$ denotes the set of all length-l paths from s to t. In this equation, the paths are exponentially damped by length to weigh the shorter paths more heavily. In addition, the Katz score between two vertices increases as they become closer. Esfandiar et. al. reported a fast and scalable method for approximating the Katz score [14].

The *hitting time*, H_{ij}, is the expected time (in terms of the number of hops) for a random walk starting from vertex i to reach vertex j [21]. The *commute time* C_{ij} is defined as the expected time for a random walk starting from vertex i to reach vertex j and return to i. That is, $C_{ij} = H_{ij} + H_{ji}$ [21]. The commute time can be calculated by the following equation.

$$C_{ij} = V \cdot \sum_{k=2}^{|V|} \frac{1}{\lambda_k} (\phi_k(i) - \phi_k(j))^2, \tag{2}$$

where λ_k and ϕ_k are k-th eigenvalue and eigenvector of the graph Laplacian [26] for given graph and V is the volume of the graph. When only a finite (usually very small) number of eigenpairs are used, the commute time is said to be *truncated*. A variant of the commute time called *scaled* commute time, SC_{ij}, is the commute time divided by the volume of the graph:

$$SC_{ij} = \frac{C_{ij}}{V}. \tag{3}$$

The commute time has been used in a wide range of applications. Saerens et al. [30] develop an application for spectral clustering based on principal component analysis of graphs, using the commute time distance. A commute time kernel based method is proposed in [15] for finding clusters, where commute time is used to measure the dissimilarities between the objects. The commute time was used even in image processing [27, 26], where the properties of the commute time metric are exploited to develop a graph-spectral method for image segmentation. More recently, Luxburg et. al. [32] showed that as the size of a given graph increases, the hitting time can be approximated by extremely simple equations that do not take into account the community structure of the graph.

4 Vertex Affinity for Community Detection

4.1 Motivation and Affinity Definition

We introduce a new affinity concept for community detection in this work. The concept of the affinity for community detection is motivated by an observation that using an improper proximity metric for solving a graph analytics often yields poor results [32] and therefore, it is critical that any proximity measure must be defined within the context of the target application to be effective. For community detection, we believe that the proximity between vertices should be gauged in terms of their *clustering strength* that is an indication of those vertices being in the same community. Measuring the clustering strength accurately and efficiently is a key challenge. The proposed vertex affinity addresses this issue.

We argue that the clustering strength between any two vertices in given graph should be high if they are connected by many short paths, because if any two vertices are in the same community, they are likely to be connected to many common vertices in the same community via short paths. We also argue that the overall clustering strength between two vertices is affected by that of those vertices on the paths connecting them. It is because if two vertices are in the same community, the vertices on the paths connecting them are also likely to be in the same community and thus should also have high clustering strength. We define the vertex affinity in a way to capture these aspects of the clustering strength.

The vertex affinity is defined as follows. Denoting the clustering strength of an edge as $ec(e(u, v))$, the the *resistance* of an edge e, denoted by $r(e)$, is defined as

$$r(e) = 1/ec(e). \tag{4}$$

Given a path $p = <v_0, v_1, \cdots, v_{k-1}>$, the *path resistance* for a path p is defined as

$$R(p) = \sum_{i=1}^{k-1} r(e(v_{i-1}, v_i)). \tag{5}$$

Let $P(s, t)$ denote the minimum independent shortest paths between s and t. The *affinity* between vertices s and t, denoted by $A(s, t)$, is defined as

$$A(s, t) = \frac{1}{\sum_{p \in P(s,t)} \frac{1}{R(p)}}. \tag{6}$$

4.2 Framework for Affinity Computation

Equations 4 through 6 above collectively define the concept of the affinity. Readers should note that we offer no rigorous definitions of the clustering strength of edges, because these definitions are highly data- and application-dependent. This also highlights the extensibility of the proposed affinity measure, as it can be easily adapted to different graphs and graph analytics applications by defining the clustering strengths accordingly. In this work, we developed a framework specifically for computing the vertex affinity for community detection. In the framework, we chose edge clustering strength measures, considering their simplicity, ease of computation, and accuracy. The proposed framework is described below in more detail.

We adopt three simple methods to quantify the clustering strength of edge in this work and refer to the quantified clustering strength of the edge as *edge clustering coefficient*. These methods are designed to be simple, ease to compute, and accurate. Furthermore, they only consider locally available information to compute the edge clustering coefficients. This not only reduces the computational overhead, but can improve the accuracy of a community detection algorithm, as vertices in a community typically have very short geodesic distance.

The first method is based on common vertex clustering coefficient that is a measure of the clustering strength of vertex [34]. We chose the vertex clustering coefficient as basis because it is easy to compute and can measure the clustering strength of a vertex with its neighbors fairly accurately. The vertex clustering coefficient of vertex v, $cc(v)$, is defined as

$$cc(v) = \frac{|\{e(u, w) : u, w \in N_v, e(u, w) \in E\}|}{d_v \cdot (d_v - 1)}. \tag{7}$$

We simply take the average of the vertex clustering coefficients of the endpoints of an edge as its edge clustering coefficient. That is, the clustering coefficient of an edge $e(u, v)$, $ecc(e(u, v))$ $= \frac{cc(u)+cc(v)}{2}$. We call this approach the `Average` method.

The next edge clustering coefficient calculation method is called `Joint` method. Here, the edge clustering coefficient $e(u, v)$ is defined as

$$ecc(e(u, v)) = \frac{|(x, y)|x, y \in J, e(x, y) \in E|}{|J| \cdot (|J| - 1)}, \tag{8}$$

where $J = N_u \cup N_v$ is the union of the vertices adjacent to either u or v. That is, in the `Joint` method the edge clustering coefficient of an edge $e(u, v)$ is defined as the density of a subgraph formed by the vertices that are adjacent to either u or v.

The last approach we adopt for the computation of the edge clustering coefficient is what we call `Triangular` method and was proposed in [28]. Basically, the `Triangular` method calculates the clustering coefficient of an edge as the ratio of the number of triangles that contain given edge to the number of all possible triangles that can be constructed on given edge. More formally, the edge cluster coefficient $ecc(e(u,v))$ is

$$ecc(e(u,v)) = \frac{z_{u,v} + 1}{min(d_u, d_v)}, \tag{9}$$

where $z_{u,v}$ is the number of triangles that contain the edge e. We discuss the computation of the affinity in more detail in following.

Assuming that each edge in given graph is assigned its edge clustering coefficient, its inverse, called edge *resistance* is assigned to each edge as its weight. This is necessary to ensure that the affinity between two vertices decreases numerically as their clustering strength between them increases. Once the resistance of each edge is determined, then given graph is reduced to a network of resistors, each of which is a measure of the affinity. The affinity between any two vertices can be modeled as the total resistance between them as shown in Equation (6). It can be possibly computed by Kirchoff's circuit laws, but a heuristic approach is adopted for its computation in our framework, because solving the Kirchoff's laws is computationally expensive, and more importantly, this heuristic approach tends to limit the ill effect of remote affinity values.

We use the minimum independent shortest paths for computing vertex affinities. Finding minimum independent shortest paths is NP-hard, so we use a greedy method to solve it. In this heuristic, given two vertices s and t, the shortest path with minimum path resistance is found by the Dijkstra's algorithm [10]. Then, all the vertices on this path except s and t are marked invalid. The Dijkstra's algorithm is invoked again to find next shortest path that consists of only valid vertices is found. This process is repeated until there exists no path connecting s and t. This set of independent shortest paths conceptually forms a combination of series and parallel resistance circuits, whose total resistance that represents $A(s,t)$ can be computed by simple circuit theory formulas as specified in Equations (5) and (6). Pseudo codes for the framework is given in Algorithm 1.

The strengths of the vertex affinity and its computation framework are as follows.

1. Since the vertex affinity measures the proximity between vertices in terms of their clustering strength, it provides proximity values that are highly relevant to the community detection, in contrast to other existing metrics. Therefore, any community detection algorithms that take an input graph with affinity-based weights can find community structures more accurately.

2. The affinity measure is quite extensible and flexible. It is because the proposed affinity and its computation framework allow the use of any edge weights that can be optimized for intended applicaitons. Such extensibility and flexibility enable the affinity measures to be applied to solving a wide range of graph analytics problems.

3. The computation of the affinity values is relatively inexpensive compared to other metrics, as they can be computed via simple heuristic approach.

Algorithm 1 Algorithm for Affinity Computation

1: Input: Graph $G = (V, E)$, where V and E are a set of vertices and edges, and two vertices s and t such that s, $t \in V$

2: Output: Affinity between s and t, $A(s, t)$

3: **for** $\forall v \in V$ **do**

  3.1: Compute clustering coefficient $cc(v) = \frac{|\{e(u,w):u,w \in N_v, e(u,w) \in E\}|}{d_v \cdot (d_v - 1)}$, where N_v and d_v denote its adjacent vertices and degree, respectively

  end for

4: **for** $\forall e = (u, v) \in E$ **do**

  4.1: Compute edge clustering coefficient $ecc(e)$

  4.2: Compute edge resistance $r(e) = ecc(e)^{-1}$

  end for

5: $P = \emptyset$

6: Find the shortest path $p = <s, v_1, \ldots, v_{k-2}, t>$ using Dijkstra's algorithm

7: **while** $p \neq$ **null do**

  7.1: $P = P \cup p$

  7.2: $\forall v \in p$ such that $v \neq s, t$, $V = V - v$

  7.3: Find the shortest path $p = <s, v_1, \ldots, v_{k-2}, t>$ using Dijkstra's algorithm

  end while

8: $l = 0$

9: **for each** $p_i \in P$ **do**

  9.1: Let $p_i = <v_0, v_1, \ldots, v_{k-2}, v_{k-1}>$, where $v_0 = s$ and $v_{k-1} = t$

  9.2: Compute $R(p_i) = \sum_{j=1}^{k-1} r((v_{j-1}, v_j))$, where $(v_{j-1}, v_j) \in E$

  9.3: $l = l + R(p_i)^{-1}$

  end while

10: $A(s, t) = l^{-1}$

5 Experimental Results

We study and discuss the properties of the affinity measure in comparison with those of other existing vertex proximity metrics in this section. These experiments also highlight its effectiveness for community detection. In most experiments, we use the simple `Average` method to compute the affinity values, because it correctly captures the characteristics of the vertex affinity with low computational overhead.

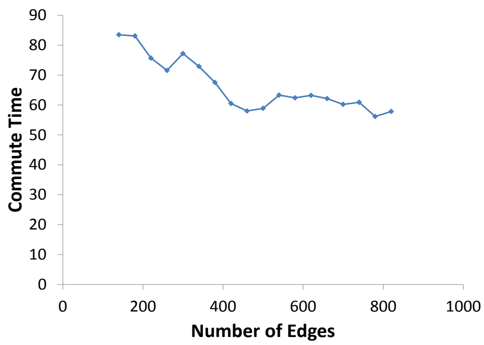
Figures 1 and 2 show the impact of the number of edges on different proximity measures. Figure 1 shows how the number of edges in a community (i.e., the density) impact various proximity measures. Here, we measure the proximity between a pair of randomly selected vertices for varying number of edges in a community that consists of 30 vertices. In measuring the Katz score, we use $\alpha = 0.05$. It is obvious that the clustering strength of two vertices in the same community should increase as the density of the community increases. Therefore, the proximity should increase or decrease monotonously, depending on how the proximity is quantified, as the density of the community increases. This can be clearly seen in Figure 1, where the values of the scaled commute time, Katz score, and our affinity measure monotonously vary as the number of edges in the community increases. However, the plot for the commute time is not monotonic. This is because the added edges in the community can increase the possibility of a random walk to diverge, depending on which vertices are connected by the new edges, while increasing the volume of the community. On the other hand, the scaled commute time decreases monotonously as the size of clique increases (as shown in Figure 2.b), because its computation is independent of the volume of graph.

This shortcoming of the commute time metric becomes more evident in Figure 2, which shows the proximity values between any two vertices in cliques of varying size for different proximity measures. Here, the proximity values are normalized in order to compared them on the same scale. As the figure shows, the commute time plot monotonously increases as the size of clique increases, indicating the weakening of the clustering strength between vertices in the clique. This is because the computation of the commute time is affected by the volume of the community. On the other hand, other proximity measures, including the affinity measure, capture the increases in the clustering strength successfully. Such dependency on the volume of graph prevents the commute time metric from correctly measuring the clustering strength and makes it unsuitable for community detection.

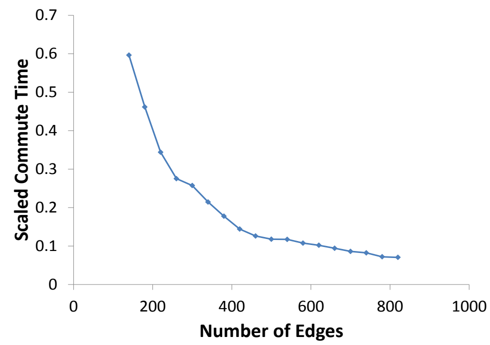
Tables 1 and 2 show the sensitivity of various proximity measures to high degree vertices in given graph, called *hubs*. In this experiment, we first construct a synthetic scale-free graph with $|V| = 500$ and $|E| = 1000$ using preferential attachment graph generation method [5, 36]. Then, we add necessary edges to form the 3- and 4-cliques. We measure the proximity from one of the vertices in the cliques via different proximity measures. Tables 1 and 2 list 10 closest vertices to the source for 3- and 4-clique test cases, respectively.

An ideal proximity measure for community detection should find the vertices in the same clique to be the closest to the source vertex. However, all existing proximity metrics considered in this experiment fail to place all vertices in the same cliques on top of the list. Rather, they determine some of the remote hubs closer to the source vertex than actual member vertices in the cliques. In contrast, the affinity measure correctly identifies the members as the closest to the source, as it is relatively insensitive to remote hubs with low clustering strength, an ideal property for community detection.

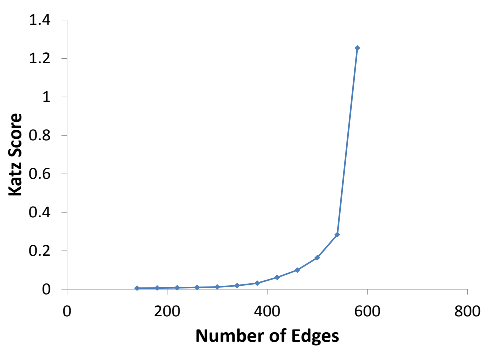
Figure 3 depicts a simple graph, in which two vertices (vertices 1 and 2) are connected by



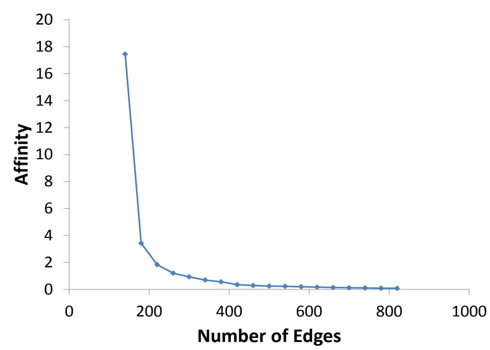
(a) Commute Time



(b) Scaled Commute Time



(c) Katz Score



(d) Affinity

Figure 1: The impact of density in a community with 30 vertices on different proximity measures. Proximity between a pair of randomly selected vertices is measured. The density of the given community increases as we add more edges to the community. For the Katz score, we use $\alpha = 0.05$.

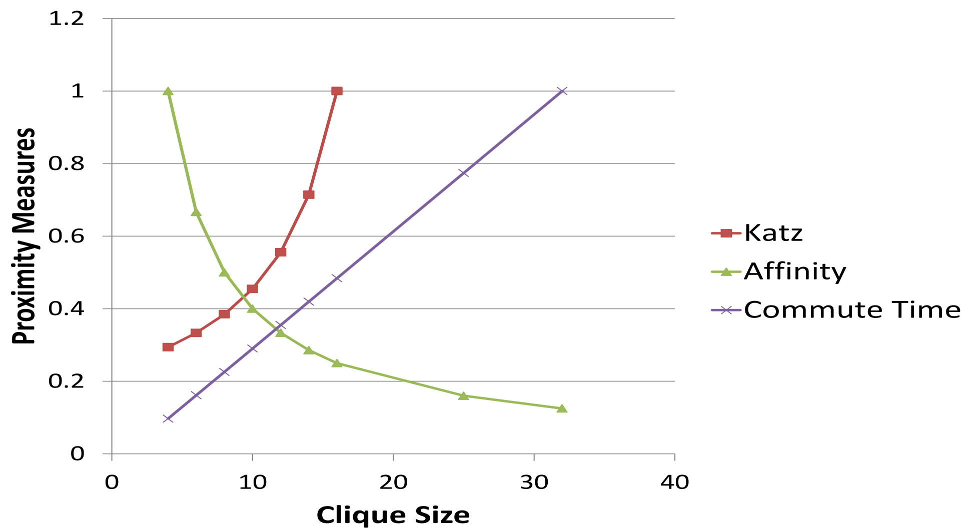


Figure 2: The proximity between vertices in cliques of varying size for different proximity measures. Proximity between any pair of vertices in each cliques is measured. The results are normalized here to compare the proximity values in the same scale. For the Katz score, we use $\alpha = 0.05$.

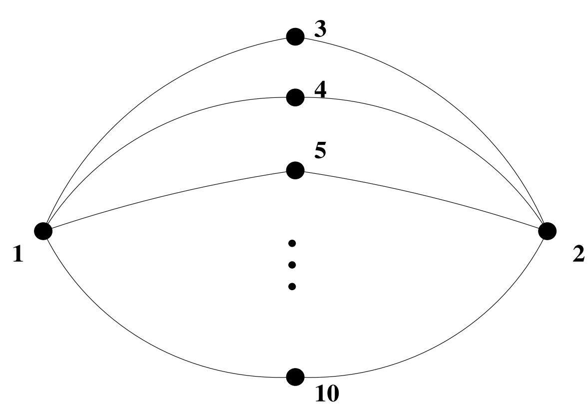


Figure 3: A synthetic graph where vertices 1 and 2 are connected by a set of independent paths to test the sensitivity of various proximity measures to in-community edges.

Table 1: Ten closest vertices to vertex 499 in a 3-clique that contains three vertices, 499, 37, and 431. For the Katz score, we use $\alpha = 0.05$. We use 5 eigenpairs in the computation of the truncated commute time.

Commute Time (CT)	Truncated CT	Katz Score	Affinity
499	**499**	5	**499**
37	**431**	**37**	**431**
5	60	**431**	**37**
2	368	**499**	5
4	420	4	500
8	136	2	4
7	42	3	2
3	104	1	392
16	15	20	275
431	398	23	220

a set of independent paths of length 2. We measure the proximity from vertex 1 to the rest of the vertices in the graph using different proximity measures. The results are presented in Table 3. Although two endpoints are well connected by a fairly large number of paths, they do not form a community, as there are no edges between all intermediate vertices (vertices 3 through 10 in Figure 3). The entries for the proximity metrics in the Table 3 contain certain proximity values indicating positive clustering strength between vertices, whereas the values for affinity measure are all infinite. Although these proximity values may indicate low clustering strength, they can introduce noises, making the community detection difficult. On the other hand, the affinity measure can divide vertices that are not in the same community cleanly and enhance the accuracy of the community detection algorithms. The truncated commute time exhibits particularly poor performance. This is a well-known problem with this metric, due to the fact that the proximity value varies to a great extent depending on the number of eigenpairs used.

Figure 4.a depicts a simple graph, where two communities (3- and 4-cliques) are connected by a path of length 3. We measure the clustering strength between vertices 1 and 9, and vertices 1 and 8 by using different proximity measures. Table 4 presents the results. Since both vertices 8 and 9 are not in the same community as vertex 1, the measured clustering strength should be very low. As can be seen in the table, the affinity measure captures this and correctly assigns infinity as the affinity from the vertex 1 to the target vertices.

In Figure 4.b, we modify the graph by adding a vertex and two edges on the path connecting two communities to form an additional 3-clique. We repeat the same experiment for this modified graph and present the results in Table 5. Readers should note that now the affinity between vertices 1 and 9 has become much smaller, as the clustering strength between vertex 1 and all other vertices that are not in the same community has increased by the newly added 3-clique. It should also be noted that the affinity among the vertices in the clique A is considerably smaller than the other vertices in the graph. On the other hand, the added edges in fact increase the commute time from vertex 1 to other vertices, because the added 3-clique increases the possibility of divergence in random walk. Furthermore, the new 3-clique has little impact on the scaled commute time and the Katz score.

Table 2: Ten closest vertices to vertex 499 in a 4-clique that contains three vertices, 499, 37, 431, and 471. For the Katz score, we use $\alpha = 0.05$. We use 5 eigenpairs in the computation of the truncated commute time.

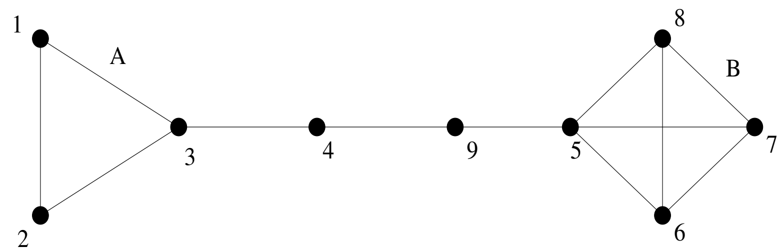
Commute Time (CT)	Truncated CT	Katz Score	Affinity
499	**499**	**37**	**499**
37	**431**	5	**431**
5	60	**431**	**471**
2	368	**471**	**37**
431	420	**499**	5
4	136	4	4
8	42	2	500
7	104	3	2
3	15	1	392
471	398	23	275

Table 3: The proximity from vertex 1 to all vertices in the graph shown in Figure 3. For the Katz score, we use $\alpha = 0.15$. We use 5 eigenpairs in the computation of the truncated commute time.

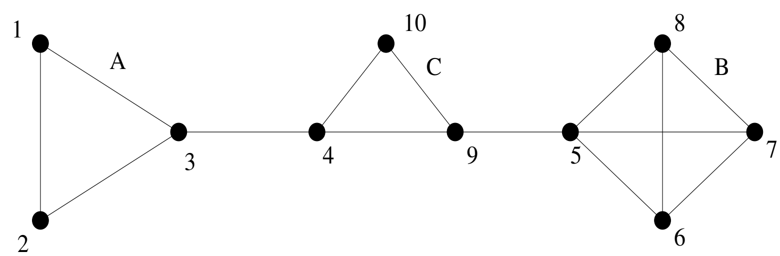
Vertex	Commute Time (CT)	Truncated CT	Katz Score	Affinity
2	8	0	0.2813	
3	18	12.0124	0.2344	
4	18	3.6581	0.2344	
5	18	10.3902	0.2344	
6	18	11.3084	0.2344	∞
7	18	11.4198	0.2344	
8	18	7.2989	0.2344	
9	18	4.7359	0.2344	
10	18	3.1763	0.2344	

Table 4: The proximity between vertex 1 to vertices 9 and 8 in the graph shown in Figure 4.a. For the Katz score, we use $\alpha = 0.15$.

Proximity Measures	Proximity(1, 9)	Proximity(1, 8)
Commute Time	64	100
Katz Score	0.0045	0.0002
Affinity	∞	∞



(a) Two cliques connected by a path of length 3



(b) Three cliques connected by two edges

Figure 4: Synthetic test graphs for evaluating the effect of overlapping communities on various proximity measures.

Table 5: The proximity between vertex 1 to vertices 9 and 8 in the graph shown in Figure 4.b. For the Katz score, we use $\alpha = 0.15$.

Proximity Measures	Proximity(1, 9)	Proximity(1, 8)
Commute Time	65.3333	107.3333
Katz Score	0.0055	0.0002
Affinity	7.5	11.23333333

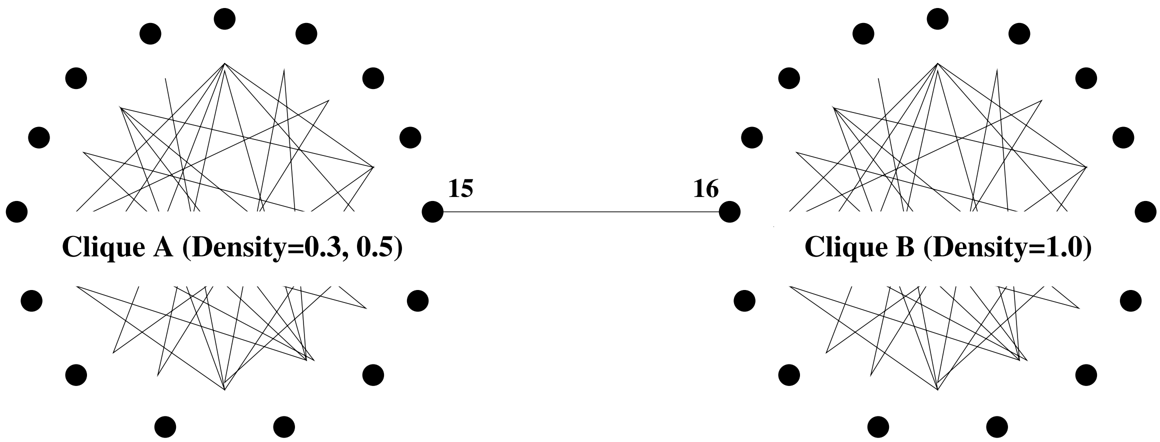


Figure 5: The construction of two quasi-cliques with certain density connected by a single edge. Clique A is built using two densities, 0.3 and 0.5. Clique B is a fully-connected clique (i.e., density = 1.0).

In Figure 5, we consider a graph that is comprised of two cliques each with 15 vertices, which are connected by a single edge ($e(15, 16)$). The clique A is a quasi-clique with certain density and the clique B is a fully-connected clique. Two density values, 0.3 and 0.5, are used in constructing the clique A in this study. We measure the clustering strength from the left endpoint of the connecting edge (vertex 15) using the same proximity measures considered in previous experiment. Because there is only one edge that is not part of any communities and vertex 15 has higher clustering strength with those vertices in the same clique, the majority of vertices that are close to vertex 15 should be in the clique A.

Table 6 lists top 15 vertices closest to the vertex 15. As shown in the table, all the proximity metrics include remote vertices in the list for low density (0.3). The Katz score show particularly poor performance, because there are more paths available to the vertices in the clique B. For higher density (0.5), however, all the metrics correctly identify the closest vertices. Table 7 presents 15 closest vertices to vertex 15 in terms of the affinity for two different edge clustering coefficient calculation methods: `Average` and `Triangular` methods.

The first column in the Table 7 shows that with the `Average` method, remote vertices (in clique B) can be selected as close vertices the affinity with Average method can choose remote vertices over the members. This occurs in the example graph because the weight on the connecting edge is positive with the Average method and the clique B has higher density than the clique A. This potentially ill aspect of the affinity can be easily corrected by utilizing the `Triangular` method. With such adaptation, the affinity measure detects the closest vertices correctly, regardless of the density of the clique A.

6 Conclusions and Future Work

A novel proximity concept called affinity is proposed for community detection (vertex clustering) in this paper. The affinity quantifies the proximity between vertices in terms of their clustering strength. A framework that combines simple graph searches and resistance circuit formulas for the efficient computation of the affinity is also developed in this work. We empirically

Table 6: List of vertices that are close to vertex 15 in Figure 5 for various proximity metrics. The vertices are listed in the order of their proximity to the vertex with measured proximity values in parenthesis. We use $\alpha = 0.05$ for computing the Katz score.

Density	Commute Time	Katz Score
0.3	15 (0.0000)	16 (0.056309)
	10 (122.6651)	10 (0.056195)
	1 (122.8442)	1 (0.053901)
	7 (135.6231)	7 (0.053636)
	11 (156.6409)	11 (0.051099)
	2 (158.6890)	15 (0.013557)
	5 (213.8228)	17 (0.008044)
	3 (226.5083)	25 (0.008044)
	12 (246.4515)	23 (0.008044)
	16 (258.0000)	22 (0.008044)
	14 (266.5115)	24 (0.008044)
	9 (288.0511)	27 (0.008044)
	22 (292.4000)	19 (0.008044)
	25 (292.4000)	18 (0.008044)
	21 (292.4000)	28 (0.008044)
0.5	15 (0.00)	7 (0.0640)
	10 (83.21)	10 (0.0634)
	7 (85.06)	11 (0.0630)
	12 (87.08)	12 (0.0612)
	11 (88.32)	1 (0.0596)
	1 (93.92)	16 (0.0567)
	8 (94.09)	13 (0.0551)
	4 (102.41)	15 (0.0211)
	3 (103.32)	8 (0.0175)
	5 (107.61)	4 (0.0146)
	14 (108.69)	5 (0.0142)
	2 (110.54)	3 (0.0124)
	6 (111.76)	14 (0.0119)
	9 (118.67)	2 (0.0094)
	13 (139.10)	6 (0.0092)

Table 7: List of vertices that are close to vertex 15 in Figure 5 for the proposed affinity measure. The vertices are listed in the order of their affinity to the vertex with measured affinity values in parenthesis.

Density = 0.3		Density = 0.5	
Average	Triangular	Average	Triangular
15 (0.0000)	15 (0.0000)	15 (0.0000)	15 (0.0000)
16 (1.8750)	10 (0.5455)	7 (0.7041)	10 (0.4327)
10 (1.9209)	7 (0.5882)	11 (0.7240)	7 (0.4511)
7 (2.2875)	1 (0.7362)	10 (0.7527)	11 (0.4757)
1 (2.7020)	2 (0.9783)	12 (0.7589)	8 (0.5119)
19 (2.9464)	13 (1.0000)	8 (0.7919)	12 (0.5185)
17 (2.9464)	11 (1.1200)	5 (0.8447)	4 (0.5687)
18 (2.9464)	5 (1.2592)	4 (0.8667)	5 (0.5703)
21 (2.9464)	3 (1.3333)	1 (0.8742)	13 (0.5806)
20 (2.9464)	12 (1.5036)	3 (0.8856)	3 (0.5920)
22 (2.9464)	9 (1.8421)	14 (0.8862)	14 (0.5938)
23 (2.9464)	14 (1.9697)	2 (0.9624)	1 (0.6006)
24 (2.9464)	6 (2.2105)	9 (0.9730)	2 (0.6452)
25 (2.9464)	8 (2.3158)	6 (0.9832)	6 (0.6682)
26 (2.9464)	4 (2.4000)	13 (1.1368)	9 (0.6746)

studied the properties of the new affinity measure and compare it to other existing proximity metrics reported in the literature. Our results show that the new affinity measure possesses properties that are essential for capturing inter- and intra-community clustering strength and can potentially enhance the performance of any community detection algorithms.

The most obvious extension of this work is to apply the proposed affinity concept to existing community detection algorithms. Input to existing community detection algorithms is typically unweighted graphs. We believe that reweighing a given graph by taking the affinity between the endpoints of each edge as the weight of the edge will greatly improve the performance of the community detection algorithms. Further, the fact that the affinity measure can assess the clustering strength of any two non-adjacent vertices motivates us to develop a new affinity-based community detection algorithm.

References

[1] B. Adamcsek, G. Palla, I. J. Farkas, I. Derényi, and T. Vicsek. Cfinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, 2006.

[2] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25:211–230, 2003.

[3] D. Aldous and J. A. Fill. Reversible markov chains, 1994.

[4] R. Andersen, D. F. Gleich, and V. Mirrokni. Overlapping clusters for distributed computation. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 273–282, New York, NY, USA, 2012. ACM.

[5] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

[6] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of community hierarchies in large networks. *CoRR*, abs/0803.0476, 2008.

[7] J. Bouttier, P. Di Francesco, and E. Guitter. Geodesic distance in planar graphs. *Nuclear Physics B*, 663, 2003.

[8] D. Chakrabarti. Autopart: parameter-free graph partitioning and outlier detection. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 112–124, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[9] A. Clauset. Finding local community structure in networks. *Physical Review E*, 72:026132, 2005.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[11] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics*, Sept. 2005.

[12] I. Derényi, G. Palla, and T. Vicsek. Clique Percolation in Random Networks. *Physical Review Letters*, 94(16):160202–+, Apr. 2005.

[13] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72:027104, Jan. 2005.

[14] P. Esfandiar, F. Bonchi, D. F. Gleich, C. Greif, L. V. S. Lakshmanan, and B.-W. On. Fast katz and commuters: Efficient estimation of social relatedness in large networks. In *WAW*, pages 132–145, 2010.

[15] F. Fouss, A. Pirotte, J. michel Renders, and M. Saerens. A novel way of computing dissimilarities between nodes of a graph, with application to collaborative filtering, 2004.

[16] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci.*, 99:7821, 2002.

[17] S. Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10), October 2010.

[18] R. Guimera, M. Sales-Pardo, and L. A. N. Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70:025101, 2004.

[19] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, March 1953.

[20] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58:1019–1031, May 2007.

[21] L. Lovsz. Random walks on graphs: A survey, 1993.

[22] M. E. J. Newman. Clustering and preferential attachment in growing networks. *PHYS.REV.E*, 64:025102, 2001.

[23] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb. 2004.

[24] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1999.

[25] P. Pons and M. Latapy. Computing communities in large networks using random walks. *J. of Graph Alg. and App. bf*, 10:284–293, 2004.

[26] H. Qiu and E. R. Hancock. Commute times for graph spectral clustering. In *CAIP*, pages 128–136, 2005.

[27] H. Qiu and E. R. Hancock. Image segmentation using commute times. In *BMVC*, pages 929–938, 2005.

[28] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proc. Natl. Acad. Sci.*, 101:2658, 2004.

[29] J. Reichardt and S. Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74:016110, 2006.

[30] M. Saerens, F. Fouss, L. Yen, and P. Dupont. The principal components analysis of a graph, and its relationships to spectral clustering. In *Proceedings of the 15th European Conference on Machine Learning (ECML 2004). Lecture Notes in Artificial Intelligence*, pages 371–383. Springer-Verlag, 2004.

[31] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA, 1986.

[32] U. von Luxburg, A. Radl, and M. Hein. Hitting times, commute distances and the spectral gap for large random geometric graphs. *CoRR*, abs/1003.1266, 2010.

[33] J. Wang, M. Li, H. Wang, and Y. Pan. Identification of essential proteins based on edge clustering coefficient. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 9(4):1070–1080, July 2012.

[34] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 4 June 1998.

[35] L. Yen, F. Fouss, C. Decaestecker, P. Francq, and M. Saerens. Graph nodes clustering based on the commute-time kernel. In *Proceedings of the 11th Pacific-Asia conference on Advances in knowledge discovery and data mining*, PAKDD'07, pages 1037–1045, Berlin, Heidelberg, 2007. Springer-Verlag.

[36] A. Yoo and K. Henderson. Parallel massive scale-free graph generators. In *Proc. SC2006*, 2006.

[37] D. Zhou and B. Schlkopf. Learning from labeled and unlabeled data using random walks. In *Pattern Recognition, Proceedings of the 26th DAGM Symposium*, pages 237–244. Springer, 2004.